MiceTrap: Scalable Traffic Engineering of Datacenter Mice Flows using OpenFlow

Ramona Trestian and Gabriel-Miro Muntean
Performance Engineering Laboratory
School of Electronic Engineering
Dublin City University, Dublin, Ireland
{ramona, munteang}@eeng.dcu.ie

Kostas Katrinis
IBM Research - Ireland
IBM Technology Campus
Damastown Industrial Estate, Dublin, Ireland
katrinisk@ie.ibm.com

***Abstract*— Datacenter network topologies are inherently built with enough redundancy to offer multiple paths between pairs of end hosts for increased flexibility and resilience. On top, traffic engineering (TE) methods are needed to utilize the abundance of bisection bandwidth efficiently. Previously proposed TE approaches differentiate between long-lived flows (elephant flows) and short-lived flows (mice flows), using dedicated traffic management techniques to handle elephant flows, while treating mice flows with baseline routing methods. We show through an example that such an approach can cause congestion to short-lived (but not necessarily less critical) flows. To overcome this, we propose MiceTrap, an OpenFlow-based TE approach targeting datacenter mice flows. MiceTrap employs scalability against the number of mice flows through flow aggregation, together with a software-configurable weighted routing algorithm that offers improved load balancing for mice flows.**

***Index Terms*—Software-defined Networks, OpenFlow, Datacenter Networks, Traffic Engineering, Routing**

I. INTRODUCTION

The current networking environment is suffering a dramatic change as new and more complex technologies are taking over. The rapid growth in cloud computing and the demand for massive-scale datacenters increases the need for more intelligent and efficient network management systems. To simplify matters, vendors and service providers started designing solutions based on the concept of Software Defined Networks (SDN) and their implementation-related technologies (e.g., OpenFlow (OF) [1]). The seminal idea of SDN is to separate the control from the data plane. Among other advantages, this separation lays the ground for fine-grained, adaptive traffic management solutions, as opposed to fixed approaches like Equal-Cost Multi-Path (ECMP) [2] or Valiant Load Balancing (VLB) [3].

For the purpose of scalable and cost-efficient traffic engineering (TE) in datacenters, traffic flows are typically classified as either short-lived (mice flows) or throughput-bound (elephant flows). Studies [4] on live datacenter traffic show that elephant flows account for less than 10% of all flows, but they carry more than 80% of the entire traffic volume. Thus, many of the proposed datacenter traffic management solutions cope with elephant flows only, while baseline routing methods like ECMP manage the mice flows. Dividing the bandwidth equally among flows is far from optimal for deadline-constrained flows like the mice flows. This could lead to waste of bandwidth and loss in terms of revenue for operators [5]. Moreover, variation in link utilization leads to hot-spots, while at the same time other links may be underutilized [6]. The latter could be used to offload traffic from the hot-spot links, thus avoiding congestion.

Because of the short lifetime of mice flows, if considered individually, they might not be of concern; however as they represent 90% of the flows [4], on ensemble they could have an important impact. In this context, applying TE solutions on 10% of the flows in a datacenter only, may not be very effective [4].

A motivational example is illustrated in Fig. 1 that depicts part of a datacenter and an application scenario, where a number of senders in Rack 1, initiate 1000 (partially) temporally overlapping short flows to the same destination in Rack 2. This many-to-one communication pattern is very common in datacenters [7], caused for example by NoSQL or distributed data processing applications. Assume that port *"In1"* at *Switch1* is already handling a good amount of traffic (e.g., due to elephant flows), exhibiting a port utilization of 96%. By employing ECMP, the traffic is equally split across the two existing paths, i.e. 500 mice flows will be routed through the already highly-loaded switch port. Assuming 1Gbps links, if the aggregate rate required by the 500 flows exceeds the residual bandwidth available at port *"In1"* (40Mbps), congestion occurs, leading to degraded applications' quality of service that generate/consume the affected mice flows. This example clearly showcases that preferential elephant flow scheduling, together with naive ECMP can be detrimental to applications inducing short flows that temporally overlap over paths with at least one common switch. To overcome this deficiency, we propose **MiceTrap, an OpenFlow-based TE approach that employs mice flow aggregation together with a weighted routing**. The idea behind this weighted routing algorithm is to find a set of dynamically computed ratios (weights) which are then used to spread the traffic at each hop across the available next hops for given traffic demands. The algorithm achieves load balancing by spreading the traffic across multiple paths.

II. RELATED WORKS

Existing multi-path routing techniques like ECMP and VLB randomly split flows across the available equal cost paths. Randomly spreading the traffic over multiple paths, without considering uneven flow sizes, can lead to transient congestion on some links. Recently, researchers have proposed traffic

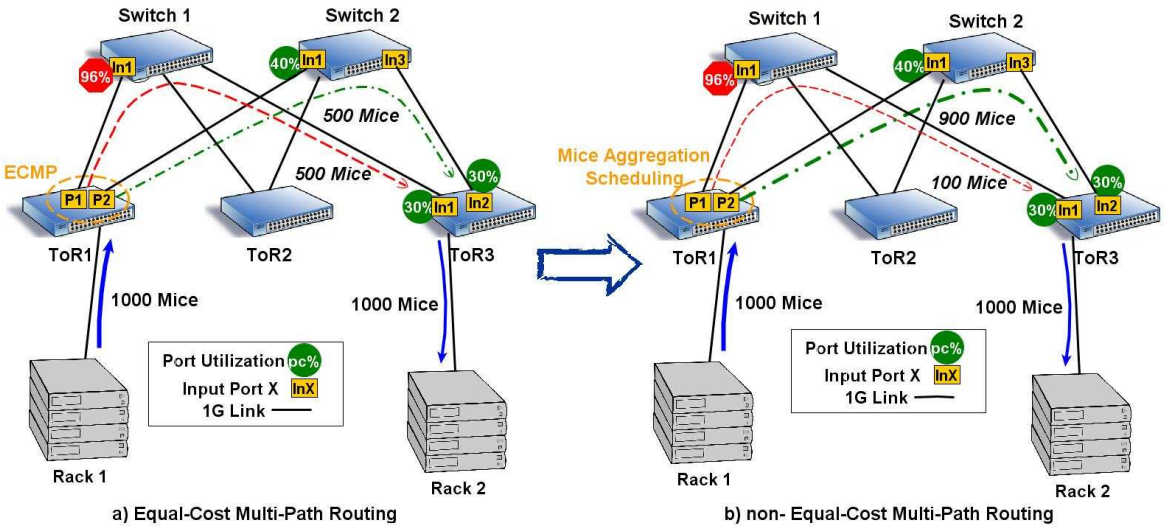


Fig. 1. Motivational Example: a) Equal-Cost Multi-Path Routing and b) non-Equal-Cost Multi-Path Routing which avoids the congestion incident

management solutions that rely on a centralized controller for route configuration, mostly leveraging on OpenFlow for switch state maintenance and traffic statistics gathering. For example, DevoFlow [8], Hedera [9], Mahout [10], all provide flow management solutions that differentiate between mice and elephant flows. The detection of elephant flows is done either at the edge switch (e.g., DevoFlow, Hedera), or at the end-host (e.g., Mahout). In both cases, a threshold for the transferred bytes is used (1-10MB for DevoFlow, 10% of NIC bandwidth for Hedera and 100KB for Mahout). When the threshold is reached, the flow is marked as an elephant flow.

Particularly to handling mice flows, Hedera uses ECMP for short-lived flows, while the elephant flows (>100MB) are handled by the OF controller only. The authors in [11] showed that Hedera performs comparable to ECMP for a traffic matrix with most of its entries corresponding to flows with less than 100MB of data. DevoFlow uses static multipath routing and the microflow path is randomly selected according to a pre-computed probability distribution. Mahout uses a static load balancing scheme without involving the controller.

In summary - and to the best of our knowledge - none of the existing datacenter TE/flow-scheduling approaches provide for dynamic, congestion-aware management of mice flows.

III. MiceTrap Architecture

Fig. 2 illustrates the MiceTrap architecture comprising of an *end-host-based elephant flow detection* mechanism module, multi-path forwarding of flow aggregates implemented on *OF switches* using standard OF techniques and custom *OF controller* modules that manage mice aggregation and routing.

A. Elephant Flow Detection

MiceTrap addresses the scheduling of mice flows and therefore it requires a mechanism to differentiate mice from elephant flows. As previously mentioned, the problem of elephant flow detection has been well researched and various solutions are available. MiceTrap employs elephant detection and marking at the end-host by using an existing kernel-level shim layer approach [10]. The mechanism makes use of a shim layer integrated in the end-host that monitors TCP socket buffers. The shim layer identifies and marks the flow as an elephant when the number of bytes in the buffer exceeds a predefined rate threshold over a given time window. The elephant flows are handled by the elephant flow scheduling scheme that works in tandem with MiceTrap, while the latter handles all unmarked flows (mice). Upon the detection of the elephant flows, a default mode is defined which handles the flows per default means (e.g. using wild-carded OF entries and using multi-path routing with ECMP). One possible technique to elevate the mice flow treatment from default to MiceTrap, could be to define a threshold over a time window and when the volume of flows targeting a specific destination rack, exceeds this threshold, MiceTrap is triggered. The mice flows are handled in a manner that collectively improves network congestion/application performance.

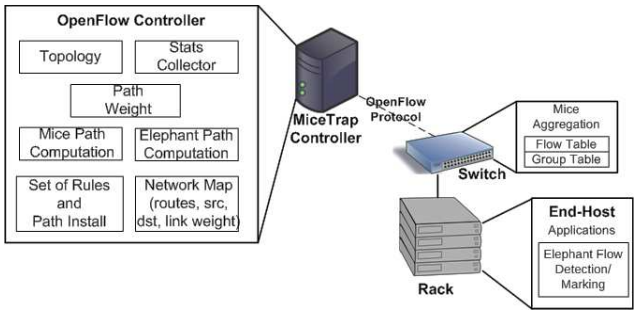


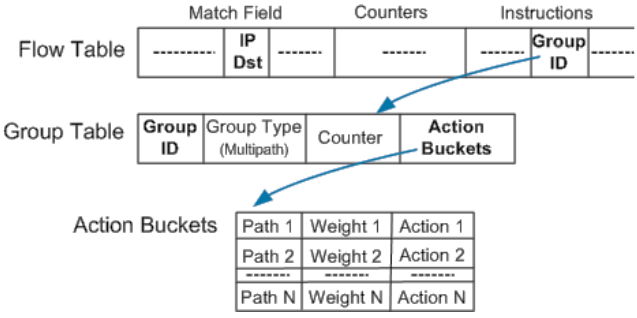
Fig. 2 MiceTrap Architecture

B. Mice Flow Aggregation

Due to the large number of mice flows in a datacenter, it is prohibitively expensive to maintain an exact rule in the switch forwarding table for each mouse flow separately. Additionally, this would hinder the scalability of traffic management due to: a) massive traffic matrix sizes that need to be handled by the controller routing module and b) inability of the limited bandwidth of the switch CPU to cope with periodically reporting a huge number of flow-entry counters to the controller. In response to this, the Mice Aggregation module

aggregates incoming mice flows per target (e.g., specific destination IP address or destination rack), thus reducing the number of rules required to apply traffic management to mice flows. This is in fact accomplished by making use of the features exposed by the OF standard only.

The OF Specification Version 1.1 proposes the use of group tables along with the flow tables to support multi-path routing. Multi-path routing is enabled by adding the ability to a flow to point to a group. Each group is composed of a set of group action buckets, and each group bucket contains a set of actions to be applied to matching flows. Each bucket carries a weight field that defines the bucket's share of the traffic processed by the group. An example is illustrated in Fig. 3. All the incoming mice flows that match the flow-entry destination IP are pointed to a group. The group table contains the action buckets with each bucket corresponding to a possible path the flow may take to reach its destination.



C. MiceTrap Controller

The MiceTrap Controller consists of: (1) *Topology block* - stores information about all the links currently up in the network; (2) *Stats Collector block* - keeps track of the links load by periodically collecting switch ports' load information; (3) *Path Weight block* - computes the paths. (4) *Mice Path Computation block* – computes the set of shortest paths between any source-destination pair; (5) *Elephant Flow Path Computation block* - computes the path for the elephant flows; (6) *Set of Rules and Path Installation block* - sends the rules to the switch; (7) *Network Map block* – network state information (e.g., current traffic matrix, established routes, etc.).

IV. MiceTrap Weighted Routing Algorithm

In order to take full advantage of the datacenter network bisection bandwidth available, as well as to protect mice flows, MiceTrap routes mice flow aggregates using a weighted multi-path routing algorithm. The routing algorithm spreads the aggregated flows across multiple links based on dynamically computed ratios in order to balance the load.

Assume the network topology is represented by a connected graph $G = (V, E)$, where V is the set of nodes and E represents the directed set of edges. Given the set of source-destination flows $F \subset V^2$, a set $N_{(s,d)}$ of shortest paths between any source-destination pairs $(s, d) \in F$ is computed. Assume that for any node $v \in V$, any path $i \in N_{(s,d)}$ has a number of M_i subpaths and each subpath $j \in M_i$ has a number of S_{ij} segments. Denoting with λ_{kji} and c_{kji} the traffic link load (taken form switch port counters) and link capacity on segment k, of subpath j, of path i, respectively, the *Link Utilisation Ratio (LUR)* is given by Eq. 1. The link load is reported by polling the respective switch port using standard OF mechanisms.

$$LUR_{kji} = \frac{\lambda_{kji}}{c_{kji}} \qquad (1)$$

At each node $v \in V$, and for any path the flow has to take to reach the destination, a weight is computed for any path $i \in N_{(s,d)}$ as given in Eq 2, with $w_i \in [0,1]$ and $\sum_i w_i = 1$. The highest the path weight, the less loaded the path is. Once computed, the path weight is updated in the action bucket weight field of each path within a certain group. Based on this, a hash function on a packet is defined in the switch in order to distribute the flows across the multiple weighted paths by sending the packets appertaining to the same flow on the same path, avoiding the re-ordering problem.

$$w_i = 1 - \frac{(\sum_j \sum_k LUR_{kji})/M_i}{\sum_i (\sum_j \sum_k LUR_{kji})/M_i} \qquad (2)$$

The pseudo code of the MiceTrap weighted routing algorithm is outlined in Algorithm 1.

Algorithm 1: MiceTrap weighted routing algorithm

Data: Network Topology G=(V,E)
Source-Destination pairs (s,d) $\in F \subset V^2$ $\forall s \neq d$
Set of shortest paths $N_{(s,d)}$ between any source-destination pairs (s, d)
Traffic load on segment k, of subpath j of path i - λ_{kji}
Link capacity of segment k, of subpath j of path i - c_{kji}
The number of subpaths of any path $i \in N_{(s,d)}$ - M_i
The number of segments of any subpath $j \in M_i$ of any path $i \in N_{(s,d)}$ - S_{ij}

Compute Link Utilisation Ratio (LUR_{kji}) of each segment
for i := 1 to $N_{(s,d)}$ **do**
  for j := 1 to M_i **do**
    for k := 1 to S_{ij} **do**
      $LUR_{kji} = \frac{\lambda_{kji}}{c_{kji}}$; /* link utilisation ratio
      k++
    end
    j++
  end
  i++
end

Compute Path Utilisation Ratio (PUR_i) of each path
for i := 1 to $N_{(s,d)}$ **do**
  for j := 1 to M_i **do**
    for k := 1 to S_{ij} **do**
      $SU_{ji} += LUR_{kji}$; /* subpath j utilisation *(SU)*
      k++
    end
    $PUR_i += SU_{ji}/M_i$; /* path utilisation ratio
    j++
  end
  i++
end

Compute Path Weight w_i of each path
for i := 1 to $N_{(s,d)}$ **do**
  $TNL += PUR_i$; **/*** total node load *(TNL)* of $v \in V$
  i++
end
for i := 1 to $N_{(s,d)}$ **do**
  $w_i = 1 - PUR_i/TNL$; /* path weight
  i++
end

V. MiceTrap Benefits

A. Forwarding State Reduction

Maintaining a rule in the forwarding table of the switch for every incoming flow is very expensive, given that switch

memory is a scarce resource. We propose to install rules matching on destination address, as the many-to-one traffic pattern is very common in datacenters for applications like MapReduce and web search [7]. For example, if 1000 mice flows arrive at a Top of Rack (ToR) switch, pointing to the same destination IP or ToR , the controller will install a single rule matching the destination, instead of having a rule for each flow (e.g., 1000 rules in the switch). Moreover, by using the group functionality of OF, whenever changes in traffic distribution occur, a single explicit group message can update a set of flow entries avoiding sending an explicit message for each flow. This way, MiceTrap saves also bandwidth along the switch-controller channel.

B. Multipath Routing

ECMP routing evenly splits the traffic across all the available next hops along the set of shortest paths to achieve fair load balancing. However, even if the traffic is equally distributed, it may not always achieve optimal load balancing.

Consider the example in Fig. 4 where B is sending 0.5Gbps of data to destination D. When A wants to send 1Gbps of data to the same destination D, it has two possible paths. Assuming that the links between switches have 1.5Gbps residual bandwidth capacity, by using ECMP the traffic from A to D will be equally spread across the two paths. However as B is already sending 0.5Gbps traffic to the same destination, the common links will be utilized at 66% of their capacity. This situation can be avoided by employing the MiceTrap weighted routing algorithm. The weights for each path are computed using Eq. 1 and Eq. 2. As Fig. 4 shows, the proposed weighted routing algorithm achieves better traffic balancing than conventional ECMP, reducing the load on the common links by 25%.

VI. Conclusions and Future Work

This paper proposed MiceTrap, a scalable scheme for traffic engineering of mice flows in a datacenter networks. Our work is motivated by the fact that management of mice flows that is oblivious to network state can lead to suboptimal utilization of datacenter fabric resources and eventually penalize short-lived flows in favor of elephant flows. This becomes even more important, when one recognizes that the sharing of network resources should be according to flow value (e.g. as quantified through SLA violations) and not necessarily based on flow size solely. To this end, MiceTrap leverages on the OpenFlow group option to handle multiple mice flows as a single forwarding aggregate and then spreads flows within an aggregate via multiple paths, using a weighted routing algorithm that takes current network load into consideration. We are currently working on prototyping our approach for the purpose of proof-of-concept and evaluation.

Acknowledgment

This work was supported by the Irish Research Council for Science, Engineering and Technology (IRCSET) through the Enterprise Partnership Scheme and the Industrial Development Agency (IDA) Ireland.

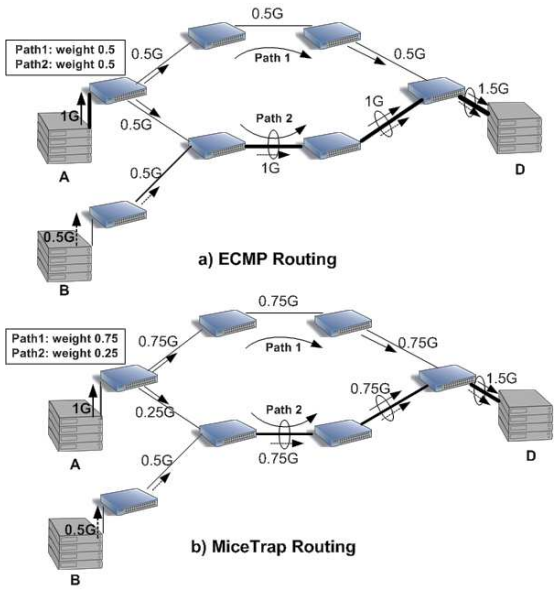


Fig. 4 Traffic Load Distribution - Example

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," ACM SIGCOMM Comput. Commun. Rev., vol. 38, no. 2, pp. 69–74, 2008.

[2] C. E. Hopps, "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992 [Online]. Available: http://tools.ietf.org /html/rfc2992.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," ACM SIGCOMM, 2009.

[4] K. Srikanth, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken. "The Nature of Data Center Traffic: Measurements & Analysis", in Proc. ACM SIGCOMM on Internet Measurement Conference (IMC), 2009.

[5] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing Flows Quickly with Preemptive Scheduling", ACM SIGCOMM, 2012.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in Proc. ACM SIGCOMM on Internet Measurement Conference (IMC), 2010.

[7] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in Data Center Networks", in ACM CoNEXT, 2010

[8] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee, "DevoFlow: cost-effective flow management for high performance enterprise networks", in Proc. ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets), 2010.

[9] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for datacenter networks", in NSDI, pp. 281-296, USENIX Association, 2010.

[10] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection", in IEEE INFOCOM, 2011.

[11] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: fine grained traffic engineering for data centers", in ACM CoNEXT, 2011.